An Experimental Analysis of PaaS Users Parameters on Applications Energy Consumption

David Guyon, Anne-Cécile Orgerie, Christine Morin

An Experimental Analysis of PaaS Users Parameters on Applications Energy Consumption

David Guyon*, Anne-Cécile Orgerie* and Christine Morin*
* Univ Rennes, Inria, CNRS, IRISA, Rennes, France – Email: {david.guyon, anne-cecile.orgerie, christine.morin}@irisa.fr

***Abstract*—Reducing the energy consumed by datacenters becomes of major importance due to the current climate changes and the increasing success of cloud computing. Studies to optimize the energy consumption of cloud systems exist but do not take into account the end-user. The goal of this paper is to understand and quantify the link between the PaaS parameters a user can configure and the application energy consumption. In this work we summarize the parameters available at the PaaS layer and measure their influence on the energy consumed by RUBiS, a web application benchmark. Different database technologies and programming languages are compared as well as their software versions. Experimentation results show that by themselves the existing PaaS parameters can variate the application energy consumption. Although it shows that different programming languages do not consume the same, we discovered that a wider energy difference exists between database technologies which means higher energy savings is possible when the less consuming technology is used. Thus, users could optimize the energy impact of their applications by carefully configuring on-hand PaaS parameters.**

***Keywords*-Cloud computing; Platform-as-a-Service; web applications; energy consumption**

I. Introduction

In 2012, the IT sector represented an annual energy consumption greater than the consumption of a country such as Russia [1]. This energy consumption continues to increase because of the success of cloud computing. The worst case scenario for 2030 shows a worrying 13% of global electricity only to power data centers [2]. Even if the current trend is going toward renewable energy sources, most of the energy consumed still comes from fossil sources [1]. On a global scale, in 2007 it represented 2% of the global GreenHouse Gas (GHG) emissions, a level of emissions similar to the global aviation sector [3]. Due to the current climate changes, reducing the energy consumption of cloud computing becomes of major importance.

Many studies [4] on green cloud computing have focused on the Infrastructure-as-a-Service (IaaS) layer which delivers virtual computing resources. Through different types of optimization, they manage to reduce the energy consumption caused by the IaaS layer. The reason why research works target IaaS clouds is because this layer is the closest to the hardware which makes it more feasible to achieve higher energy savings. On top of the IaaS layer stands the Platform-as-a-Service (PaaS) layer that delivers a development and execution environment for user's applications. This layer experiences growing success [5], however, few research works have been conducted on the possible energy optimization that can be done at this cloud service layer.

Application developers use PaaS clouds to develop, deploy and execute their applications. At each of these 3 phases, PaaS clouds deliver several parameters to control the execution of applications [6]. During the development phase, the application's programming language is specified as well as the version of the language interpreter when required. The same configuration is available for the database management system. It defines what software and their version need to be installed in the execution environment. These parameters show that PaaS users have a control over the application's execution but the monitoring feedback does not include any energy related information.

Several types of applications can be deployed on PaaS clouds. Silva et al. [7] classify different categories of applications with their own characteristics in terms of resource utilization such as, high performance computing applications, data-intensive distributed applications and transactional applications. The latter refers to typical multi-tier web applications which is the type of application selected in this study because many PaaS providers target web applications.

The goal of this work is to understand the impact of PaaS level users' decisions on the energy consumption. In this work, we focus on web applications and target the software parameters offered by PaaS clouds. In the experimental evaluation, we make real energy consumption measurements of a typical web application developed in two different programming languages and compatible with two different database technologies. Also, two language interpreters in different versions are used. Experimentation results reveal a relation between the configuration of the PaaS environment and the energy consumed by the servers on which the application is running. We also show that the software selection and also their version can have an impact on the application energy consumption of up to 12% increase. The impact is even more significant when changing the database technology. The results show that a database technology can make the database tier consume 141% more energy than with another technology.

The remaining of this paper is organized as follows. In Section II we present the context and motivation of this work. Section III details our methodology. Experimental results are discussed in Section IV. Section V discusses the feasibility of our measurement approach. Finally Section VI concludes.

II. Context and Motivation

Despite the fact that the exact definition of PaaS is still an open debate, Giessmann and Stanoevska give this definition [8]: "*Platform as a Service refers to an execution environment, wherein external developers deploy and run their complementary components.*". This type of cloud is dedicated to application developers because it provides online resources to build and run their applications without the need to download or install anything [9]. Applications run in VMs managed by the PaaS provider. This management includes the maintenance of the operating system and installed software stack.

This study focuses on a specific type of application widely deployed in PaaS clouds: multi-tier web applications. This type of application has a fluctuating resource utilization based on the request load (workload). Web applications are commonly separated in three tiers [10]: the *web server* delivers web pages with a web cache system, the *application server* executes the application's logic and the *database server* stores the data. This separation offers elasticity: cloud resources can be added or removed on each tier to adapt the performance to the request load. It avoids high response time, unappreciated by web site users. In this work we use a type of application inspired by 3-tiered architectures. We separate the database from the logic as it is a usual behavior in PaaS clouds. However, the separation of the web server and the application tiers is not a by default setup and remains complex to realize. In our case, the web server and the application server features execute together in the application tier. To summarize, our study targets 2-tiers web applications decomposed in a web server/application tier and a database tier.

At the time the PaaS application is ready to execute, the provider delivers an environment where the application will run. Concretely an environment corresponds to a set of VMs where a software stack is already installed. This stack depends on the details of the software project given by the developer, such as the programming language, the version of the language interpreter, the database management system and the packages requirement of her application. The available software configuration differs from one provider to another. As an example, the languages Ruby, Node.js, PHP, Python and Java are supported by three widely used PaaS providers, Engine Yard [11], Heroku [12] and Clever Cloud [13]. However, Engine Yard offers a finer control over the version of PostgreSQL than the others, Heroku can natively execute Clojure applications and Clever Cloud provides a larger choice of versions for the programming language interpreters. Each provider has its own way to implement the software management system. Some PaaS providers prefer to have a single large VM image with all software installed on it to ease its maintenance. In the open-source solution ConPaaS [14], the provider manages many VM images, each one for a specific type of environment. In such architecture a VM image is dedicated to PHP applications, while another VM image is only used to execute Java Servlet applications. A more detailed presentation of PaaS cloud providers can be found in [6].

A PaaS user cannot interact with the computing resources and the underlying operating system directly. However, as explained before, users have access to parameters to control the execution of their applications. To summarize, they have access to the following software parameters:

- software stack to use depending on the programming language of the application
- database management system to store the data
- software versions for running the application and the database

PaaS parameters allow users to undertake actions that impact the execution of their applications. Despite the availability of these parameters, it is complex for users to understand if there is a variation in energy consumption linked with the choices they make when defining their applications. This is why we want to verify if the configuration of the parameters is related to the energy consumption. The quantification of the parameters' influence on the energy consumption would deliver which of them are the more influencing. In a previous work [15], we showed that, in an IaaS cloud context, the VMs size parameter given to users have an impact on the energy consumed by the cloud. Small size VMs favor the consolidation but may have lower performance. The current study focuses on PaaS clouds that deliver additional parameters that need to be analyzed to understand their impact on the energy consumption. At this level of the cloud stack, the energy consumption depends on both the virtual resources and the deployed applications. In [16] the authors also show that there exists a relation between programming languages and energy consumption. Yet, their study is out of the context of cloud computing, the amount of computing resource is fixed and software versions are not compared. The virtualization layers of the cloud computing creates a distance between users applications and the real hardware consuming the energy. This distance increases the complexity of creating a relation between PaaS parameters and energy consumption.

In this paper, the power consumption and performance of a 2-tier web application are analyzed. The application tier exists in two different versions, each version developed in a different programming language and both versions are compatible with several versions of programming language interpreters. As for the database tier, it is also available in two different versions with two different database management system technologies. An experimental analysis is applied on the previously listed software parameters that are accessible thanks to the web application's implementation.

III. Experimental Setup

This section provides the details about the selected benchmark and workload. The PaaS cloud platform that has been used to run the experiments is presented. We also define the experimental scenarios and the metrics.

A. RUBiS Benchmark

Our experiments use a benchmark approach similar to previous studies [17]. From the existing web application benchmarks, we selected RUBiS [18], an online auction website modeled after the Internet website eBay.

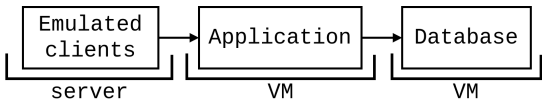

Fig. 1: Architecture of the RUBiS benchmark.

As shown in Figure 1, RUBiS is composed of 3 elements. The *client* is a Java program that simulates users navigating on the website. It is responsible for sending requests to the *application* tier and behaves according to the workload definition (detailed in the next section). The application tier handles both the web serving and the execution of the application source code. This is why the entry point of the application tier is an Apache server that serves static HTML pages or redirects users' requests to the running application when asking for dynamic content pages. In order to build the dynamic pages, the application requests data from the *database* tier. The data retrieved from the *database* is included in pages built by the *application* which are then sent to the *client*. Depending on the user's request, it may imply one or many connections between the application and the database.

Our main criterion of choice for this benchmark is its implementation in several programming languages. The application tier is available in both PHP and Java Servlet. The PHP version only needs the Apache server to execute, while the Java Servlet also requires the application server Tomcat. Thus, RUBiS can execute with a PHP interpreter or with a Java interpreter, both versions after the same features. As for the database tier, it comes with the MySQL relational database management system. We modified the RUBiS benchmark in order to support the PostgreSQL object-relational database system because this is a widely used database technology (see Section III-D).

B. Workload

RUBiS comes with a workload system that simulates users sending requests on the web pages of the emulated web application. The benchmark's configuration file offers a fine tuning of the workload. Our workload is defined to last for a total of 30 minutes, including 5 minutes at the beginning to increase the load (up ramp) and 5 minutes at the end to decrease the load (down ramp). The workload contains a total of 2000 threads where each thread represents a client doing navigation on the website. A maximum of 2000 transitions can be made by each user. All experiments conducted in this study use the workload definition presented in this section.

C. Platform-as-a-Service Cloud Layer

The benchmark executes on top of our own cloud system. As for the hardware, the servers provided by the Grid'5000 platform [19] are equipped with 16 cores from the Intel Xeon CPU E5-2620 processor, 32GB of RAM, 600GB of HDD and a 10GB Ethernet connection. The virtualization layer is handled by QEMU version 2.1.2 which is used through libvirt 1.2.9 [20], a virtualization API to simplify the management of virtual hosts.

In our experiment, the application tier and the database tier of RUBiS execute in two separate VMs. The VM operating system executes a Debian 3.16.43. The management of VMs in our PaaS cloud is inspired by ConPaaS [14]. There are several pre-configured VMs for each version of the benchmark. Each VM has the software stack required by the version of RUBiS it has to execute. These VMs represent the application tier of the RUBiS architecture presented in Figure 1. In addition to these pre-configured VMs, there are also two VMs dedicated to the database tier. A VM executes a MySQL database while the other VM executes a PostgreSQL database. Both VMs have their database tables pre-filled with the same data. As for the virtual resources, all VMs are given 2 vCPUs, 2GB of RAM and 20GB of HDD.

D. Scenarios Definition

There are two groups of scenarios: the application scenarios and the database scenarios.

An application scenario corresponds to a VM image in which the required software stack for a specific version of RUBiS is installed and ready to use. In total we designed 6 scenarios. Two scenarios are dedicated to execute the PHP version of RUBiS with respectively PHP5 and PHP7. In the case of PHP7, it is deprecated to use the *mysql* command, thus the application source code has been updated to use the advised command *mysqli*. The four remaining scenarios are for the Servlet version running with Java 7 or Java 8, and either with Tomcat 7 or Tomcat 8. Details of the scenarios are shown in Table I.

TABLE I: Application scenarios with the version of the software installed in each application VM.

scenario	version of software
PHP5	PHP 5.6.30-0+deb8u1
PHP7	PHP 7.0.20-1 dotdeb+8.2
T7J7	Tomcat 7.0.56 and OpenJDK 1.7.0
T7J8	Tomcat 7.0.56 and OpenJDK 1.8.0
T8J7	Tomcat 8.0.14 and OpenJDK 1.7.0
T8J8	Tomcat 8.0.14 and OpenJDK 1.8.0

As shown in Table II, we defined two database scenarios. The first one corresponds to a VM where the MySQL relational database management system is installed. This is the database tier delivered with RUBiS without any modification. The second scenario is a VM with the PostgreSQL object-relational database management system installed. This one has been created from scratch in order to be able to compare different database technologies. The data in the MySQL database has been exported/imported to the PostgreSQL database so that the content of each database is identical. The MySQL

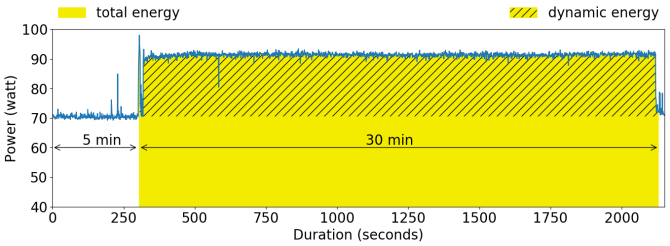

Fig. 2: Definition of the total energy consumption and dynamic energy consumption. The graph represents the power consumption of the application tier VM for the PHP5 version of RUBiS.

and PostgreSQL installed versions are the default versions available in the Debian packages repository at the time of this study. In order to use either one or the other database scenario, compatibility updates were required in the legacy RUBiS application tier. A new version of the PHP7 has been developed which uses the PHP Data Objects (PDO) extension instead of the *mysqli* command. This special version of the PHP7 scenario is only used in the experiment where the database tier changes.

TABLE II: Database scenarios with the version of the database management system installed in each database VM.

scenario	version of software
MySQL	MySQL 5.5.57
PostgreSQL	PostgreSQL 9.4.12

Apache2 is used as the HTTP server and redirects the network requests directly to the PHP application or the Java application through Tomcat. The Apache 2.4.10 version has been installed in all application VMs.

E. Energy and Performance Measurements

Two metrics are used to quantify the energy consumption and the average request response time.

The energy metric, in Watt-hour (Wh), represents the amount of energy which is consumed by the run of the workload for a given scenario. Figure 2 shows how this metric is calculated. Power measurements are possible thanks to the power meters [21] plugged to the servers used for our experiments. They deliver a power value each second at a resolution of 0.125W. A power meter measures the entire consumption of a server, this is why each VM is deployed in a different server. In the case where the deployed VM is running alone on a server, the VM energy consumption is defined as the dynamic energy consumption of the server. As shown in Figure 2, the dynamic energy (hatched area) is the total energy consumption (yellow area) minus the idle energy of the server. For this calculation, two values are required: idle and total energy consumption of the server. For the former, a 5 minutes long measurement is done before each experiment when servers do not execute anything. This preliminary measurement provides the idle power consumption of each server. As for the latter, the server power consumption is measured during the execution of the 30 minutes long workload which gives the total energy consumption.

As for the average request response time, it is used to express the system's performance. This metric, in ms, is measured internally by the benchmark and returned in the execution logs. We use the average response time that the 2000 users take to access the home page of the RUBiS website. When the workload starts, all users try to reach the home page at the same time which implies an important requests load. We observed that the response time of the home page varies significantly between scenarios because of the different software technologies used. The response time variation becomes almost negligible a few seconds after the workload starts because requests are not received at the same time. The larger variation of the response time at the start of the workload gives a better feedback on the application's performance, which is why this is the one analyzed in this work.

F. Experimental Structure

To evaluate the impact of PaaS software parameters on the energy consumption and response time of applications, two different experiments were designed.

1) varying application tier: Two PaaS parameters are taken into account: the programming language and the version of the language interpreter. This experiment follows the software configuration of each application scenario listed in Table I. The database tier remains fixed and uses the MySQL technology. Each scenario executes its application tier and database tier in fixed size VMs with 2 vCPUs. The workload considered is the one defined in Section III-B.

2) varying database tier: Only the database management system parameter is taken into consideration. This parameter can take two values as presented in Table II: MySQL and PostgreSQL. The application tier remains fixed and uses the special version of PHP7 with PDO. PDO allows the PHP7 application tier to be compatible with both database scenarios. All VMs have a fixed size of 2 vCPUs and the workload is the same as the first experiment.

IV. Analysis of Experimental Results

This section presents the results of the two previously described experiments. Each experiment result shows the mean value and the standard deviation of 5 executions. The 5 executions ran in parallel on different servers.

A. Impact of Software Versions

Figure 3 displays both the dynamic energy consumed by the application tier and the database tier when we apply the workload on each application scenario (see Table I). The graph on the right represents the average response time of all clients to access the home page (metric explained in Section III-E).

It shows that on average the application tier of the two PHP scenarios consumes 7.27% less energy compared to the four

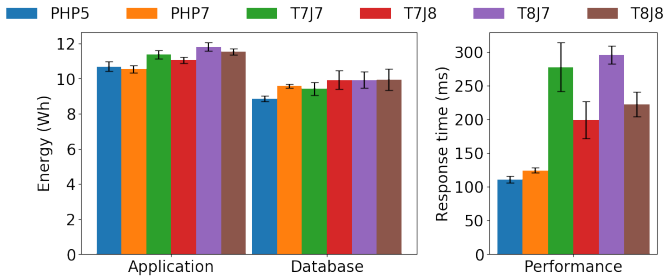

Fig. 3: Dynamic energy consumption of application and database tiers and response time for each application scenario.

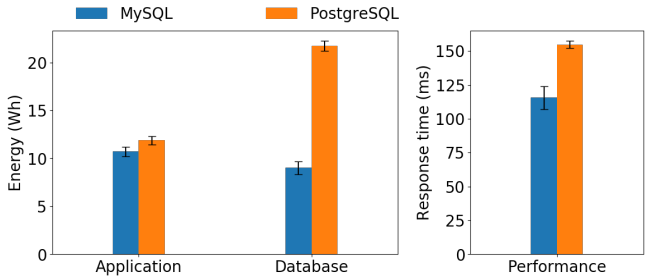

Fig. 4: Dynamic energy consumption of application and database tiers and response time for each database scenario.

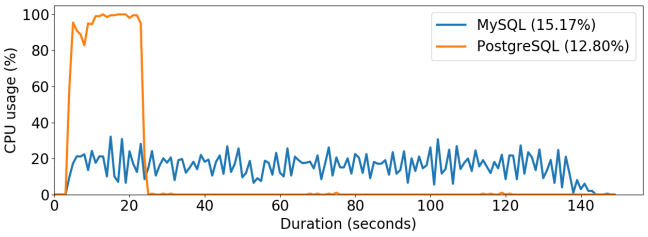

Fig. 5: CPU usage of both VM databases when processing 200 SQL requests in parallel. Each request is a SELECT of 100.000 entries from the *users* table.

Java scenarios. This difference is explained by the additional cost caused by the Java Virtual Machine in the Java versions.

Differences also exist between versions. Even if scenarios PHP5 and PHP7 have a similar application energy consumption, PHP7 presents a database consuming 8.14% more than the PHP5 database. The utilization of the database differs in these two scenarios because PHP7 is using the more recent *mysqli* command. The update from *mysql* to *mysqli* adds a security layer which implies additional computations, thus additional energy costs. However, in both PHP scenarios the response time is similar with a small variation of 13.6 ms.

Similar energy variations can be seen between Tomcat versions and Java versions. Going from Java 7 to Java 8 always present a decrease in application energy consumption: 2.91% less energy consumption when using Tomcat 7 and 2.29% with Tomcat 8. Regarding the response time, it significantly improves by 28.29% (Tomcat 7) and 24.58% (Tomcat 8) when the application programming language goes from Java 7 to Java 8. The opposite behavior appears when going from Tomcat 7 to Tomcat 8. The application energy consumption increases by 3.76% and 4.42% as well as the response time by 6.34% and 11.85% for the Java 7 and Java 8 scenarios, respectively. However, the database energy consumption remains stable.

Varying programming languages and software versions mainly impact the energy consumption of the application VM with a maximum of 12.04% more energy consumption between PHP7 and Tomcat 8 Java 7. A smaller energy impact is shown on the database VM with a maximum variation of 5.5% between scenarios Tomcat 7 Java 7 and Tomcat 8 Java 8 (here we ignore the PHP5 scenario because of its use of the deprecated *mysql* command).

B. Impact of Database Management Systems

The impact on energy and performance of the database technology is shown in Figure 4. For each database scenario (see Table II), results present the dynamic energy consumed by the application tier and the database tier as well as the average response time of all clients to access the home page (metric detailed in Section III-E)

As expected, the application tier has a similar energy consumption in both scenarios that is explained by their source code that slightly differs. Their energy consumption has a variation of 1.18Wh which represents an increase of 11%. However, the database tier shows a significant energy consumption increase of 140.84% when changing from the MySQL to the PostgreSQL database technology. In addition to this important increase in energy consumption, requests response time also increased by 33.91%. The MySQL technology is a relational database system while the PostgreSQL technology is an object-relational database system. While MySQL has been initially designed for web applications, it is not the case for PostgreSQL. Their internal mechanisms organize information differently which justifies the important energy consumption difference.

In Figure 5, a different benchmark is used to show the CPU usage of both database technologies when stressed with an identical workload. The workload is generated by 4 copies of a program sending 50 SELECT SQL requests to retrieve 100.000 entries from the *users* table. These 4 duplications of the program execute in parallel. Each database executes in a VM with 4 vCPUs, thus 100% CPU usage means that all CPUs are fully used. MySQL takes about 140 seconds to process this workload while PostgreSQL takes less than 30 seconds. This difference of durations is explained by the ability of PostgreSQL to benefit from parallelization. This is also why PostgreSQL is able to reach 100% CPU usage while MySQL barely exceeds 25% (1 vCPU used at its maximum capacity). The higher is the CPU utilization, the higher is the power consumption. In the experiment presented in Figure 4, the workload continuously stresses the database during 30 minutes. In the case of PostgreSQL, it results in a higher total

CPU utilization than MySQL, which explains the significant energy increase of 140.84%.

In comparison with the previous experiment, the database parameter has more impact on the energy consumption of the database VM than the software versions parameter on the application VM.

V. Discussion on Feasibility and Usability

We now put into context the results obtained in this work with the compatibility of our approach with other applications and how the users can benefit from these.

Other types of application: Our experimental analysis is done on a multi-tier web application because this is the most widely deployed type of application in the context of PaaS clouds. However, our work remains relevant with other types of application as long as the application programming language and software versions can be analyzed in terms of energy consumption and performance. For example, in the case of Python applications, the energy consumption and performance of a benchmark Python program need to be analyzed with different versions of the Python interpreter and with different amount of cloud resources. This way it is possible to adapt this work to other types of PaaS applications.

Preliminary benefits: As explained in [22], increasing consumer knowledge with energy-related information has the potential to change consumer behavior. Providing the correct information at the right time to users may have an impact on their development decisions. A team of developers familiar with 2 different programming languages may be incentivized by energy/performance information to decide which language to use. However, this performance/energy information needs to be known before starting to write the application source code. This preliminary information can have a significant impact on the energy consumption over the long term.

Benefits during applications execution: When an application is already written in a specific programming language, it would be too expensive in terms of time and budget to rewrite the entire source code. Yet, users can still tune parameters such as the software versions. This change may only require small updates of the source code so that the application can execute with a less consuming software version.

Measurements on users applications: In this work, the energy consumption and performance of a benchmark are analyzed while varying PaaS parameters. The measurement analysis results are valid in the case of the multi-tier web application benchmark we used. However, results may differ with users applications that are too different from the benchmark. To avoid this difference between the benchmark and real applications, a solution would be to execute continuous energy measurements on the real application that is running. This solution would present the actual real energy consumption and performance of users applications. However, this kind of measurement requires to deploy power models of VMs [23]. Beside the implementation complexity, each power model consumes energy which in total may result in additional energy cost. Real measurements may not be worth the energy cost and the limited accuracy of values given by these power models.

VI. Conclusion and Future Work

Users of PaaS clouds have access to parameters such as the programming language, the database technology and the softwares versions. However, it is difficult to know how related these parameters are with the energy consumption. In this study, we analyze and quantify the energy consumption of a web application deployed in a PaaS cloud with different configurations of parameters. The web application is available in both PHP and Java Servlet versions and its database is compatible with the MySQL and the PostgreSQL database management technologies. We compared the energy consumption and performance of each application's version.

Experimentation results show that, in the case of a 2-tier web application deployed in a PaaS cloud, the already available PaaS parameters have an impact on the energy consumption of the cloud application. Indeed, two different programming languages show different energy profiles. Applications written in Java present on average a 7.84% increase in energy consumption in comparison with PHP versions. In addition to the programming languages, a different energy consumption is found when changing the version of the software used to execute the application. The energy consumption of the application VM increases by 4.42% when moving from Tomcat 7 to Tomcat 8. Similar energy profile differences can be seen with different database management systems. The PostgreSQL database consumes 140.84% more energy than the MySQL database. These experimentation results are a first step towards a better understanding of the relation between PaaS parameters and cloud applications energy consumption.

Parameters delivered by PaaS systems also include a control over the size and number of VM instances users desire to execute their cloud application. As a future work we would like to extend this study to evaluate the impact of these parameters on the underlying energy consuming devices. In this future work we plan to measure the energy and performance of an application while we increase the size of the VM (number of vCPUs) the application is running on. We will also analyze, for an identical total amount of vCPUs, the energy and performance of duplicating the application tier in a varying number of VMs.

Acknowledgments

Experiments presented in this paper were carried out using the Grid'5000 experimental test-bed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies (see https://www.grid5000.fr).

References

[1] G. Cook, J. Lee, T. Tsai, A. Kong, J. Deans, B. Johnson, and E. Jardim, "Clicking Clean: Who is Winning the Race to Build a Green Internet?" *Greenpeace Inc., Washington, DC*, 2017.

[2] A. S. Andrae and T. Edler, "On Global Electricity Usage of Communication Technology: Trends to 2030," *Challenges*, vol. 6, no. 1, pp. 117–157, 2015.

[3] G. Cook, T. Dowdall, D. Pomerantz, and Y. Wang, "Clicking Clean: How Companies are Creating the Green Internet," *Greenpeace Inc., Washington, DC*, 2014.

[4] A.-C. Orgerie, M. D. d. Assunçao, and L. Lefèvre, "A Survey on Techniques for Improving the Energy Efficiency of Large-scale Distributed Systems," *ACM Computing Surveys*, vol. 46, no. 4, pp. 47:1–47:31, Mar. 2014.

[5] Right Scale, "State of the Cloud Report," Tech. Rep., 2015.

[6] S. Costache, D. Dib, N. Parlavantzas, and C. Morin, "Resource Management in Cloud Platform as a Service Systems: Analysis and Opportunities," *Journal of Systems and Software*, 2017.

[7] M. Silva, M. R. Hines, D. Gallo, Q. Liu, K. D. Ryu, and D. Da Silva, "CloudBench: Experiment Automation for Cloud Environments," in *Cloud Engineering (IC2E), 2013 IEEE International Conference on*. IEEE, 2013, pp. 302–311.

[8] A. Giessmann and K. Stanoevska, "Platform as a Service–A Conjoint Study on Consumers' Preferences," in *Proceedings of the International Conference on Information Systems, ICIS 2012*, F. G. Joey, Ed. AIS Electronic Library: Association for Information Systems, December 2012, p. 20.

[9] G. Lawton, "Developing Software Online with Platform-as-a-Service Technology," *Computer*, vol. 41, no. 6, 2008.

[10] D. Huang, B. He, and C. Miao, "A Survey of Resource Management in Multi-Tier Web Applications," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 3, pp. 1574–1590, 2014.

[11] "Engine Yard." [Online]. Available: http://www.engineyard.com/

[12] "Heroku." [Online]. Available: https://www.heroku.com/

[13] "CleverCloud." [Online]. Available: https://www.clever-cloud.com/

[14] G. Pierre and C. Stratan, "ConPaaS: A Platform for Hosting Elastic Cloud Applications," *IEEE Internet Computing*, vol. 16, no. 5, pp. 88–92, 2012.

[15] D. Guyon, A.-C. Orgerie, C. Morin, and D. Agarwal, "How Much Energy can Green HPC Cloud Users Save?" in *Parallel, Distributed and Network-based Processing (PDP), 2017 25th Euromicro International Conference on*. IEEE, 2017, pp. 416–420.

[16] A. Noureddine, A. Bourdon, R. Rouvoy, and L. Seinturier, "A Preliminary Study of the Impact of Software Engineering on GreenIT," in *Green and Sustainable Software (GREENS), 2012 First International Workshop on*. IEEE, 2012, pp. 21–27.

[17] T. Palit, Y. Shen, and M. Ferdman, "Demystifying Cloud Benchmarking," in *Performance Analysis of Systems and Software (ISPASS), 2016 IEEE International Symposium on*. IEEE, 2016, pp. 122–132.

[18] Rice University Bidding System, "RUBiS." [Online]. Available: http://rubis.ow2.org/

[19] "Grid'5000." [Online]. Available: https://www.grid5000.fr

[20] Red Hat, "libvirt: The Virtualization API," 2012.

[21] M. D. De Assuncao, J.-P. Gelas, L. Lefevre, and A.-C. Orgerie, "The Green Grid5000: Instrumenting and using a Grid with energy sensors," in *Remote Instrumentation for eScience and Related Aspects*. Springer, 2012, pp. 25–42.

[22] K. Tsuda, M. Uwasu, K. Hara, and Y. Fuchigami, "Approaches to induce behavioral changes with respect to electricity consumption," *Journal of Environmental Studies and Sciences*, vol. 7, no. 1, pp. 30–38, 2017.

[23] C. Mobius, W. Dargie, and A. Schill, "Power Consumption Estimation Models for Processors, Virtual Machines, and Servers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 6, pp. 1600–1614, 2014.